Programación Python para Machine Learning

Lección 10: Modelos de Ensemble.

eip ESCUELA INTERNACIONAL POSGRADOS

ÍNDICE

Lección 10: Modelos de Ensemble.

1. Introducción

Los modelos de Ensemble son estrategias de Machine Learning capaces de lograr un rendimiento excelente ante cualquier tipo de problema predictivo.

Esta capacidad les viene del carácter intrínseco que les da su naturaleza, ya que son el resultado de unir las predicciones de varios predictores. La posibilidad de poder corregir los errores cometidos por cualquier modelo a nivel individual lleva a tener una alta precisión en global.

Esta lección presenta diferentes modelos de Ensemble de predictores y cómo pueden ser implementados en Python.

Objetivos

- Presentar los principios en los que se basan los modelos de Ensemble.
- Conocer las distintas estrategias de Ensemble.
- Dominar las técnicas de implementación de los modelos de Ensemble en Python.
- Examinar los puntos clave que determinan el rendimiento de los modelos de Ensemble.

2. Principios teóricos de los Ensemble

Los modelos de Ensemble son un tipo de metodología de Machine Learning que se basa en la combinación de diferentes modelos predictivos. En este sentido, este tipo de estrategias pueden ser consideradas como meta-modelos más que un modelo en sí mismo.

La idea que subyace detrás de los métodos Ensemble es que el hecho de unir las salidas de diversos predictores traerá consigo un incremento del rendimiento, ya sea en términos de mejora de la precisión de las predicciones, o en términos de aumento de la estabilidad de la respuesta de los modelos.

Se podría hacer un agrupamiento de los modelos de Ensemble según su esquema. Por un lado, estarían los Ensemble secuenciales, cuya estrategia sería que los modelos base se generen en cadena uno tras otro, ponderando sucesivamente las instancias sobre las que las predicciones fueron erróneas en el modelo base anterior. Por otro lado, estarían los modelos paralelos, los cuales se basan en crear modelos base en paralelo y promediar los resultados en conjuntos.

Conceptos

Se denomina **modelo base** a cada uno de los modelos predictores que son combinados dentro de un método de Ensemble.

Otra dimensión en la que categorizar los Ensemble sería hacer una distinción entre métodos homogéneos y heterogéneos. En tal caso, los Ensemble homogéneos serían aquellos que tiene muchos predictores base del mismo tipo, mientras que los heterogéneos utilizan la combinación de diferentes tipos de algoritmo base.

2.1. Combinaciones simples

Antes de valorar cualquier estrategia de combinación más compleja, hay una serie de combinaciones simples de predictores que sería conveniente considerar:

- **Ensemble por promedio:** Un método de Ensemble muy rápido y fácil de podría ser el de promediar predicciones. Se trataría simplemente de entrenar por separado un determinado número de modelos base de predicción para luego combinar sus salidas mediante un promedio.
- **Ensemble por votación:** Un modelo de Ensemble por votación considera cada salida de los diferentes estimadores como un voto. Luego puede combinarlas bien mediante un sistema de reglas de mayoría, o bien calculando cuál ha sido la salida más alta de todas.

2.2. Bagging

Los métodos de Bagging son un método de Ensemble que consiste en tomar múltiples instancias con reemplazo del conjunto de datos de entrenamiento y entrenar un modelo para cada muestra. A la hora de la predicción, para calcular la salida final del Ensemble se combinan las predicciones de todos los modelos base.

Esta combinación de predicciones se realiza mediante un promedio para la regresión, o mediante votación para la clasificación, normalmente, por mayoría.

Cuando los resultados se combinan de este modo, la varianza general del modelo disminuye y presenta un mejor rendimiento como resultado.

2.3. Boosting

Los modelos de Boosting trabajan en todas las etapas con el conjunto completo de entrenamiento, y se manipulan los pesos de las instancias para generar modelos distintos.

La idea se basa en que en cada iteración se incremente el peso de las instancias mal predichas por el modelo, provocando que en el entrenamiento del siguiente predictor estos objetos tendrán una mayor importancia y tendrán más probabilidades de ser correctamente predichas. La Ilustración 1 muestra gráficamente un resumen comparativo entre las distintas fases de los métodos de Ensemble Bagging y Boosting.

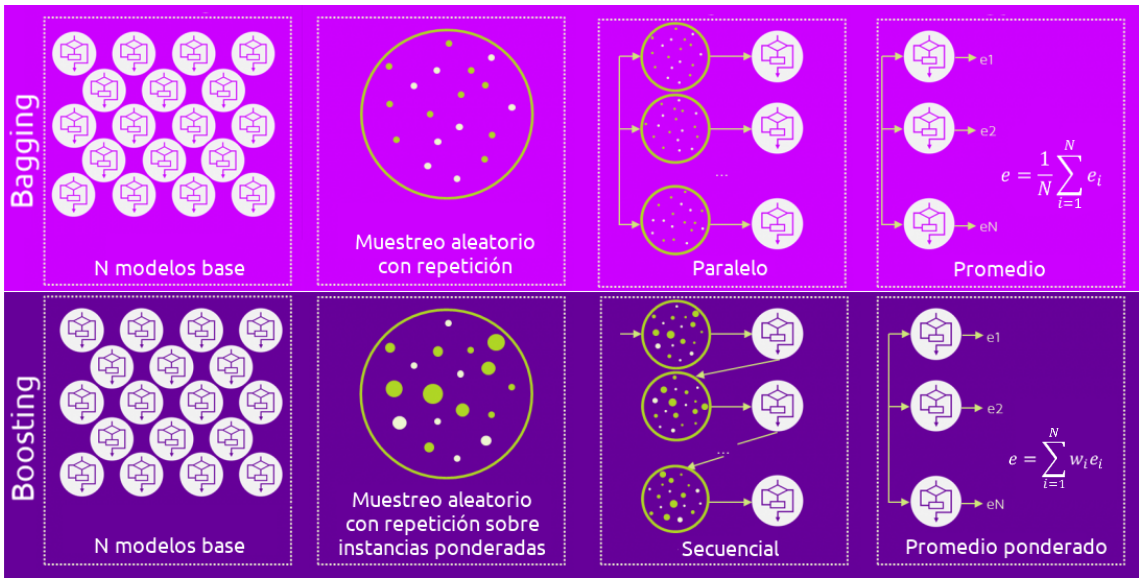

Ilustración 1: Esquema comparativo Bagging - Boosting

2.4. Stacking

Los modelos de Stacking implica el entrenamiento de un metamodelo para combinar las predicciones de los modelos base. El proceso, en una primera fase, entrena varios predictores base con los datos de entrenamiento. Posteriormente, en una segunda etapa, se entrena el modelo final con el conjunto de datos de entrenamiento considerando como características adicionales las predicciones de los modelos de la primera fase. Este enfoque suele utilizar un conjunto heterogéneo de modelos base en la primera etapa, porque la diversidad favorece el rendimiento.

3. Implementación en Python de varios modelos de Ensemble

Los modelos de Ensemble presentan una versatilidad dentro del Machine Learning que los hacen ser un modelo apropiado para resolver tanto problemas de clasificación como problemas de regresión. Esto se debe a que solo dependen del modelo base con el que sean configurados.

En Python es posible implementar diversos modelos de Ensemble utilizando el módulo `ensemble` de scikit-learn, tanto para clasificación como para regresión.

En todo caso, hay que recordar que cuando se trata con un modelo de Ensemble en scikit-learn, todas las variables deben ser de numéricas. Por tanto, toda variable categórica en el conjunto de datos debe ser convertidas en numéricas del modo más adecuado y representativo posible.

3.1. Ensemble para clasificación con Bagging

Scikit-Learn cuenta con la clase `BaggingClassifier` dentro de su módulo `ensemble`, que permite crear una estrategia de Bagging para proceder con problemas de clasificación. Por defecto, `BaggingClassifier` tiene como clasificador base la clase `DecisionTreeClassifier`, aunque es posible cambiar esta configuración utilizando el parámetro `base_estimator`.

Otros dos parámetros a tener en cuenta a la hora de configurar un modelo de Bagging son el número de clasificadores base que utilizará `n_estimators` y el número de instancias del conjunto de datos que serán muestreados para cada uno `max_samples`. Se hará uso del conjunto de datos *magic* del repositorio *UCI Machine Learning* para ilustrar el proceso.

En todo caso, hay que recordar que cuando se trata con un modelo en scikit-learn, todas las variables deben ser de numéricas.

Por tanto, toda variable categórica en el conjunto de datos debe ser convertidas en numéricas del modo más adecuado y representativo posible.

```
from pandas import read_csv
from sklearn.ensemble import BaggingClassifier
import time, random, numpy
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier

seed=random.seed(time.time())
filename = '../datasets/magic04.data'

col_names=['fLength','fWidth','fSize','fConc','fConc1','fAsym','fM3Lon
g','fM3Trans','fAlpha','fDist','class']
data = read_csv(filename, names=col_names)

X = data[data.columns[:-1]]
Y = data['class']

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size =
0.7, random_state=seed)

scaler =StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test= scaler.transform(X_test)

base_model= KNeighborsClassifier()
base_model.fit(X_train, y_train)
y_pred = base_model.predict(X_test)
acc= accuracy_score(y_test, y_pred)

model = BaggingClassifier(base_estimator=base_model, n_estimators=100,
max_samples=0.1 , random_state=seed)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
acc= accuracy_score(y_test, y_pred)
print(acc)
```

3.2. Ensemble para regresión con Bagging

Scikit-Learn cuenta con la clase `BaggingRegressor` dentro de su módulo `ensemble`, que permite crear una estrategia de Bagging para proceder con problemas de regresión. Por defecto, `BaggingRegressor` tiene como regresor base la clase `DecisionTreeRegressor`, aunque es posible cambiar esta configuración utilizando el parámetro `base_estimator`. Otros dos parámetros a tener en cuenta a la hora de configurar un modelo de Bagging son el número de regresores base que utilizará `n_estimators` y el número de instancias del conjunto de datos de entrenamiento que serán muestreados para cada uno `max_samples`. Se hará uso del conjunto de datos *magic* del repositorio *UCI Machine Learning* para ilustrar el proceso.

En todo caso, hay que recordar que cuando se trata con un modelo en scikit-learn, todas las variables deben ser de numéricas. Por tanto, toda variable categórica en el conjunto de datos debe ser convertidas en numéricas del modo más adecuado y representativo posible.

```
from pandas import read_csv
from sklearn.ensemble import BaggingRegressor
import time, random, numpy
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.neighbors import KNeighborsRegressor

seed=random.seed(time.time())
filename = '../datasets/housing.csv'

col_names=['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS',
'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV']
data = read_csv(filename, names=col_names,sep=" ")

X = data[data.columns[:-1]]
Y = data['MEDV']
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size =
0.3, random_state=seed)

scaler =StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test= scaler.transform(X_test)

base_model= KNeighborsRegressor()
base_model.fit(X_train, y_train)
y_pred = base_model.predict(X_test)
rmse= numpy.sqrt(mean_squared_error(y_test, y_pred))
print(rmse)


model = BaggingRegressor(base_estimator=base_model, n_estimators=100,
max_samples=0.1 , random_state=seed)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
rmse= numpy.sqrt(mean_squared_error(y_test, y_pred))
print(rmse)
```

3.3. Ensemble para clasificación con Boosting

Scikit-Learn cuenta con la clase `AdaBoostClassifier` dentro de su módulo `ensemble`, que permite crear una de las estrategias de Boosting más populares: AdaBoost.

Por defecto, `AdaBoostClassifier` tiene como clasificador base la clase `DecisionTreeClassifier`, aunque es posible cambiar esta configuración utilizando el parámetro `base_estimator`. Otros dos parámetros a tener en cuenta a la hora de configurar un modelo de AdaBoost son el número de clasificadores base que utilizará `n_estimators` y la tasa de aprendizaje `learning_rate`. Se hará uso del conjunto de datos *magic* del repositorio *UCI Machine Learning* para ilustrar el proceso.

En todo caso, hay que recordar que cuando se trata con un modelo en scikit-learn, todas las variables deben ser de numéricas. Por tanto, toda variable categórica en el conjunto de datos debe ser convertidas en numéricas del modo más adecuado y representativo posible.

```
from pandas import read_csv
from sklearn.ensemble import AdaBoostClassifier
import time, random, numpy
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier

seed=random.seed(time.time())
filename = '../datasets/magic04.data'

col_names=['fLength','fWidth','fSize','fConc','fConc1','fAsym','fM3Lon
g','fM3Trans','fAlpha','fDist','class']
data = read_csv(filename, names=col_names)

X = data[data.columns[:-1]]
Y = data['class']

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size =
0.7, random_state=seed)

scaler =StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test= scaler.transform(X_test)

base_model= DecisionTreeClassifier()
base_model.fit(X_train, y_train)
y_pred = base_model.predict(X_test)
acc= accuracy_score(y_test, y_pred)
print(acc)

model = AdaBoostClassifier(base_estimator=base_model, n_estimators=50)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
acc= accuracy_score(y_test, y_pred)
print(acc)
```

Conceptos

AdaBoost (Adaptative Boosting) **es una de las técnicas más populares de Boosting. AdaBoost es adaptativo en el sentido de que los modelos base posteriores se ajustan a favor de las instancias mal clasificadas por los clasificadores anteriores.**

3.4. Ensemble para regresión con Boosting

Scikit-Learn cuenta con la clase `AdaBoostRegressor` dentro de su módulo `ensemble`, que permite crear una de las estrategias de Boosting más populares: AdaBoost. Por defecto, `AdaBoostRegressor` tiene como regresor base la clase `DecisionTreeRegressor`, aunque es posible cambiar esta configuración utilizando el parámetro `base_estimator`.

Otros dos parámetros a tener en cuenta a la hora de configurar un modelo de AdaBoost son el número de regresores base que utilizará `n_estimators` y la tasa de aprendizaje `learning_rate`. Se hará uso del conjunto de datos *housing* del repositorio *UCI Machine Learning* para ilustrar el proceso.

En todo caso, hay que recordar que cuando se trata con un modelo en scikit-learn, todas las variables deben ser de numéricas. Por tanto, toda variable categórica en el conjunto de datos debe ser convertidas en numéricas del modo más adecuado y representativo posible.

```
from pandas import read_csv
from sklearn.ensemble import AdaBoostRegressor
import time, random, numpy
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.neighbors import KNeighborsRegressor

seed=random.seed(time.time())
filename = '../datasets/housing.csv'

col_names=['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS',
'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV']
data = read_csv(filename, names=col_names,sep=" ")
```

```
X = data[data.columns[:-1]]
Y = data['MEDV']

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size =
0.3, random_state=seed)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test= scaler.transform(X_test)

base_model= KNeighborsRegressor()
base_model.fit(X_train, y_train)
y_pred = base_model.predict(X_test)
rmse= numpy.sqrt(mean_squared_error(y_test, y_pred))
print(rmse)

model = AdaBoostRegressor(base_estimator=base_model, n_estimators=100)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
rmse= numpy.sqrt(mean_squared_error(y_test, y_pred))
print(rmse)
```

4. Consejos prácticos sobre los métodos de Ensemble

No se puede considerar un método de Ensemble mejor que otro de un modo absoluto. Esta cuestión es dependiente de los datos y las circunstancias.

Ante la situación de que el modelo base por sí solo tiene un bajo rendimiento, la estrategia de Bagging por lo general no obtendrá una mejor respuesta.

Por el contrario, un método Boosting podría generar un modelo con menos error, al basarse en la optimización de las predicciones mal realizadas.

Si la dificultad del modelo base es el acoplamiento excesivo, *overfitting*, la mejor opción es aplicar un modelo de Bagging. Boosting por su parte no ayuda en este sentido. De hecho, Boosting lo que trata es ese problema en sí, es decir, encontrar un mayor ajuste del modelo sobre el conjunto de datos.

5. PUNTOS CLAVE

En esta lección se ha aprendido:

- Conocer los conceptos generales de los métodos de Ensemble.
- Hacer un recorrido sobre todos los modelos de Ensemble y sus características.
- Implementar en scikit-learn un modelo de Bagging tanto para problemas de clasificación como regresión.
- Implementar en scikit-learn un modelo de Boosting tanto para problemas de clasificación como regresión.
- Profundizar en aquellos aspectos que pueden ayudar a la hora de aplicar un método de Ensemble.